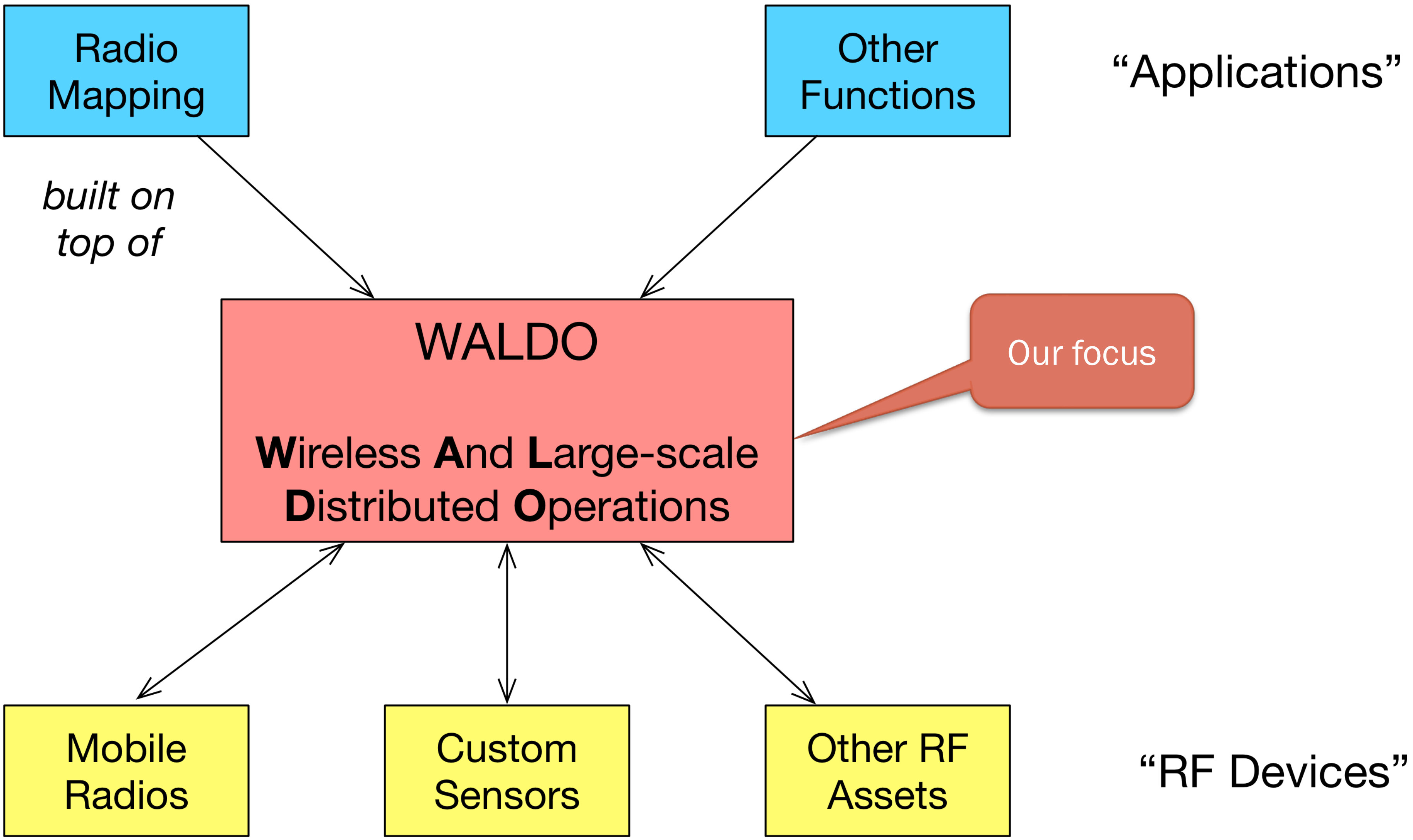
Using Standardized Semantic Technologies For Discovery And Invocation Of RF-Based Microservices

JAKUB MOSKAL
MITCH KOKAR
OLIVIER HUREZ-MARTIN

NOVEMBER 14, 2018

Context: WALDO



The Challenge

- Roles/Responsibilities:
  - RF Devices offer services, e.g. spectrum sensing, specific jamming technique, signal detection, using their own API's
  - Applications request the services offered by the RF device, but are NOT developed against any specific device/service API
  - **Middleware** matches the requests with available services, but is NOT developed for any specific device or application

- Middleware designs typically rely on common data models that all participants must be aware of:
  - Typical technologies:
    - Relational databases
    - Data exchange formats: XML, JSON, etc.
    - Well-defined, binary data structures and protocols
  - Semantics is provided only informally:
    - Accompanying documentation
    - Hardcoded in procedural code managing the data

- As a domain evolves, data models must be updated
  - Because these changes have no explicit semantics, software developed around the model must be updated, which can be difficult, expensive and time-consuming

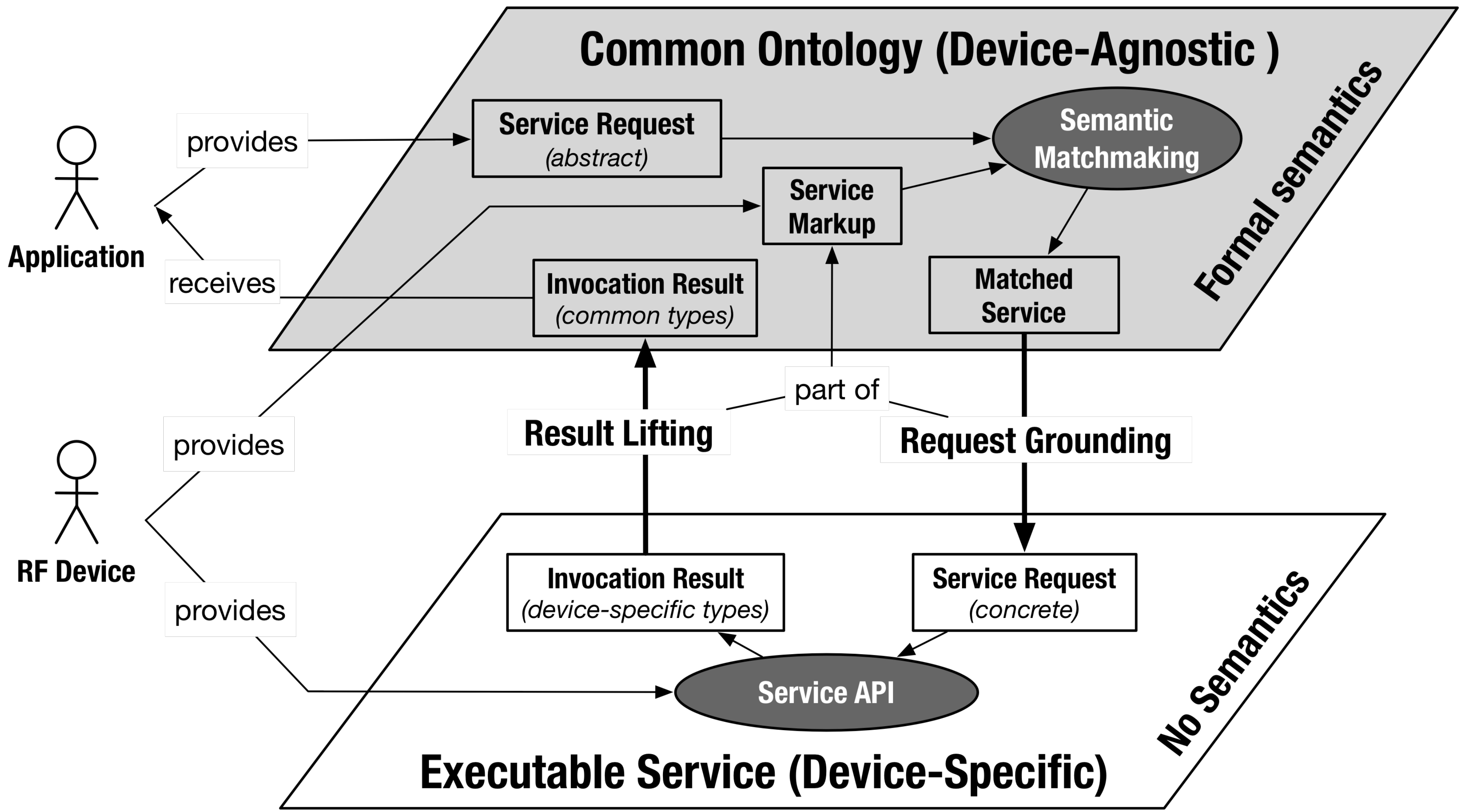
- Dedicated, built-in at design-time, application/vendor-specific extensions to these data models work as long as one has full control over all participants
  - Most often, however, they lead to loss of interoperability in the long-term

Why Semantic Technologies?

- In semantic technologies, semantics is **explicit**
- Ontologies can be dynamically extended to accommodate new terms (even at runtime!)
- They are processed by **general-purpose** inference engines to derive new, implicit facts
  - For instance, to connect the base data model with the extensions
  - As ontologies evolve, the engines remain intact

- These qualities make ontologies an ideal choice for a future-proof middleware

Conceptual Framework – Semantic Web Services



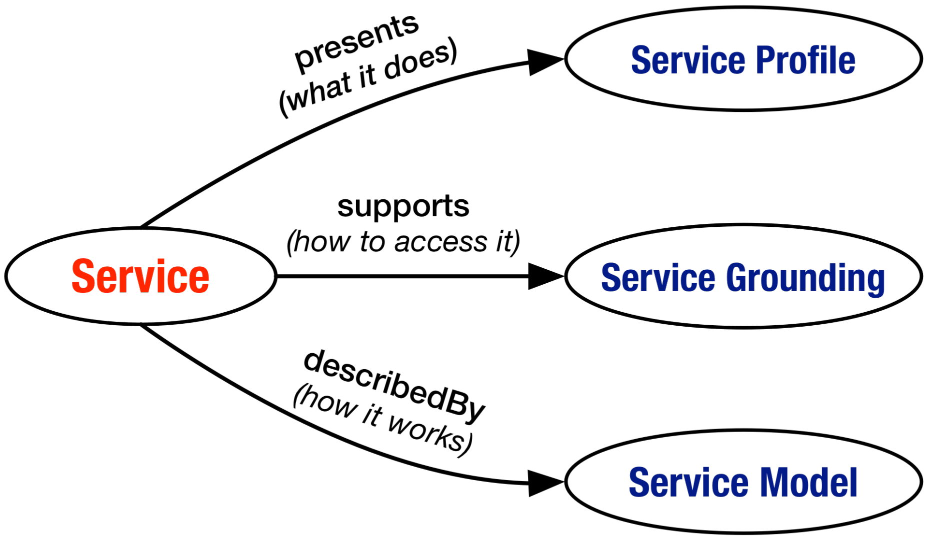
Rationale

- Matching and optimization algorithms operate at a high level of abstraction
  - No need to worry about syntactic variances, e.g. different method names or data structures for semantically equivalent capabilities
- Inference engine can help determine implicit matches
  - For instance, determine the match between spectrum sensing and energy detection
- Matching algorithms are domain-independent, hence future-proof with respect to changes in the domain
- Foundation for decomposition of services enable new possibilities:
  - Optimization, e.g. parallel or redundant execution
  - Opportunistic matching, e.g. detection of 802.11 signals could be decomposed into signal sampling, DSSS detection, OFDM detection and cyclostationary features detection
- Support for legacy applications and RF devices via automatic inference
- No strict requirements on the service execution
  - SOAP/WSDL, REST, JSON-RPC

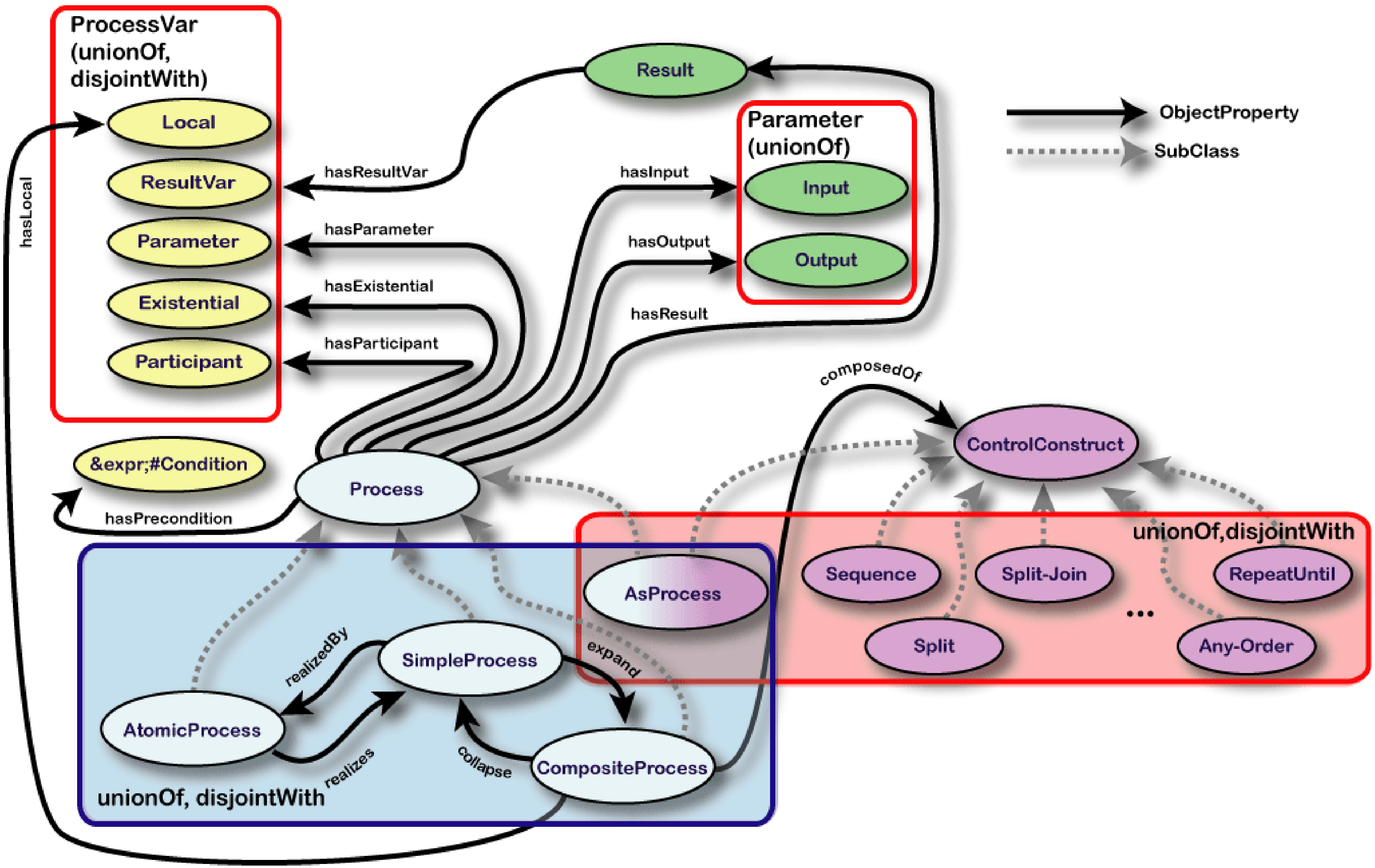
W3C OWL-S – Top View

- W3C **OWL-S** is an upper ontology to describe the semantics of services

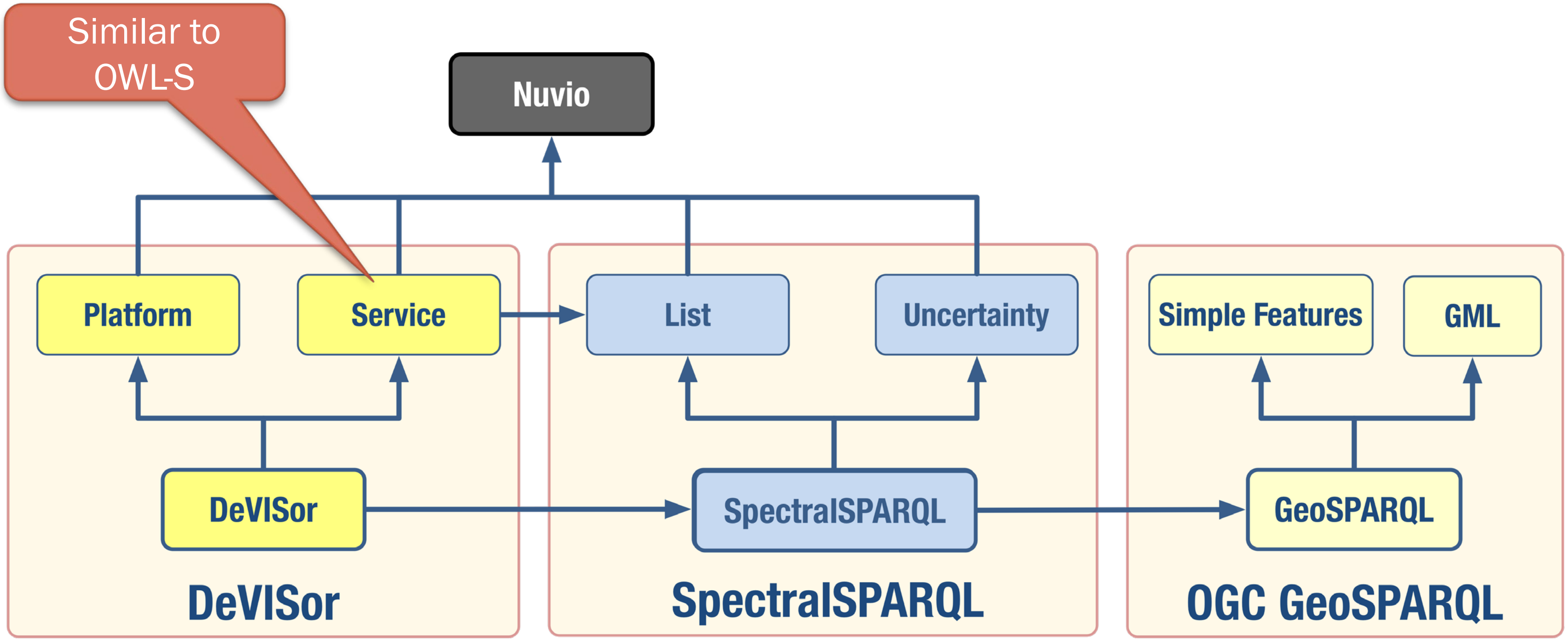
- Three main components:
  - Service Profile
    - Advertising and discovering services
    - Input, Output, Preconditions, Effects (IOPE)
  - Service Model
    - Service composition
    - Choreography and orchestration
  - Service Grounding
    - Binding between the logic-based semantic service profile, the process model and Web Service interface
    - Facilitates execution



OWL-S – Service Model

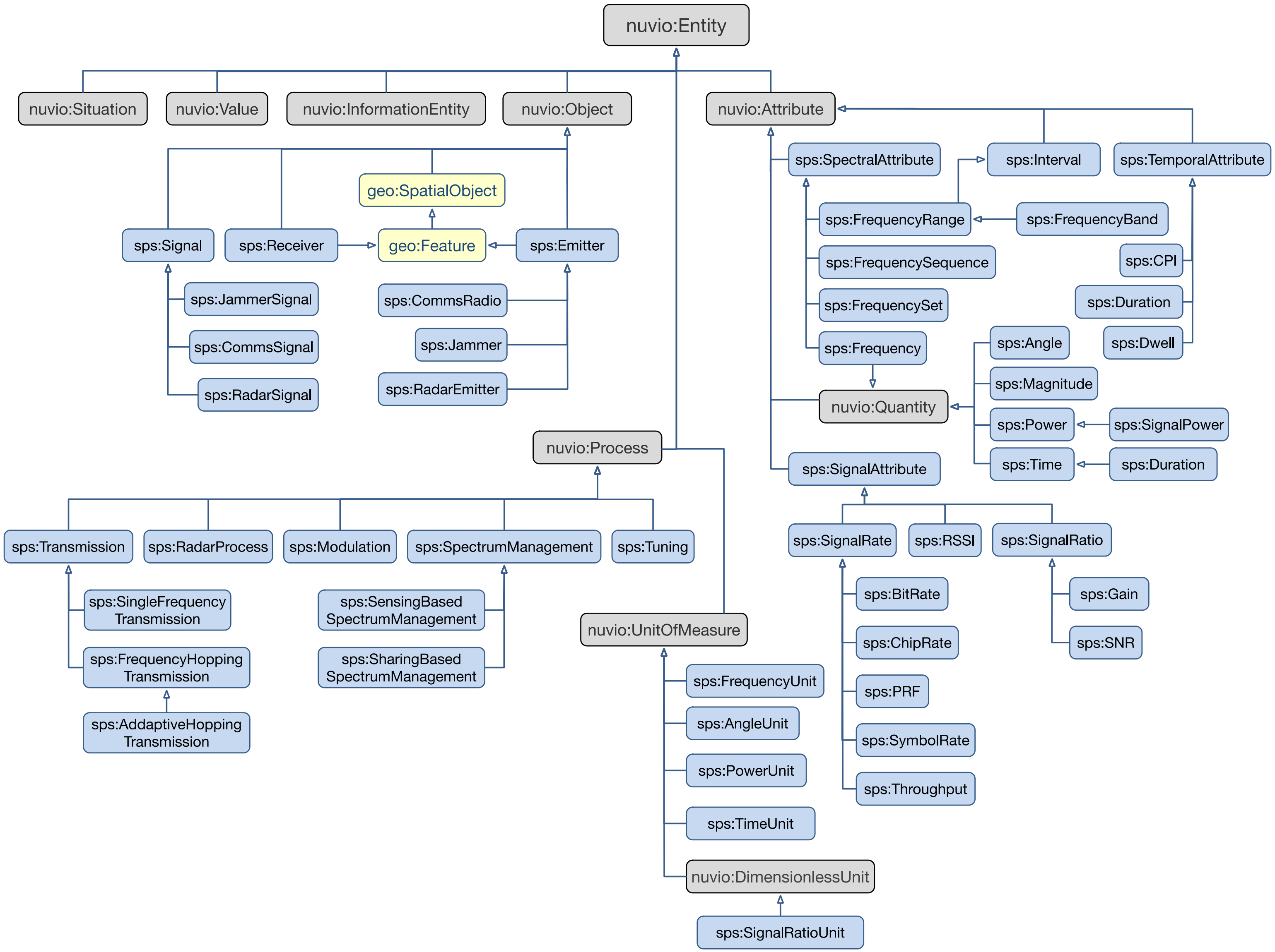


Conceptual Plane – Ontology Architecture



Moskal J. J., Kokar, M. M., Roman, V., Normoyle, R. B.,&Guseman, R. P. (2017, October). **Towards a SpectralSPARQL Standard for Exchanging EMS Knowledge**. In Military Communications Conference, MILCOM 2017 IEEE. (Restricted Paper Session)

SpectralSPARQL



Execution Plane – JSON-RPC

- REQUEST
  - jsonrpc – protocol version / optional
  - method – to be invoked
  - params – data structure to be passed as an input to the method / optional
  - id – of the request

- Examples:

```
{ "jsonrpc": "2.0",
  "method": "subtract",
  "params": {"subtrahend": 23, "minuend": 42},
  "id": 3
 }

{ "jsonrpc": "2.0",
  "method": "subtract",
  "params": [42, 23],
  "id": 3
 }
```

- RESPONSE
  - jsonrpc – protocol version /optional
  - result – data structure representing the output of the method invocation
  - error – passed if there was an error
  - id – of the request

- Examples:
  - Result:

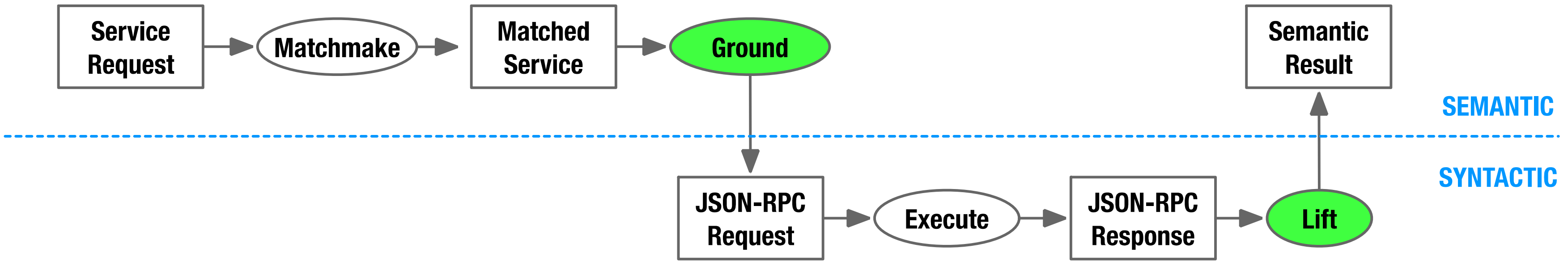
```
{"jsonrpc": "2.0",
 "result": 19,
 "id": 3
}
```

  - Error:

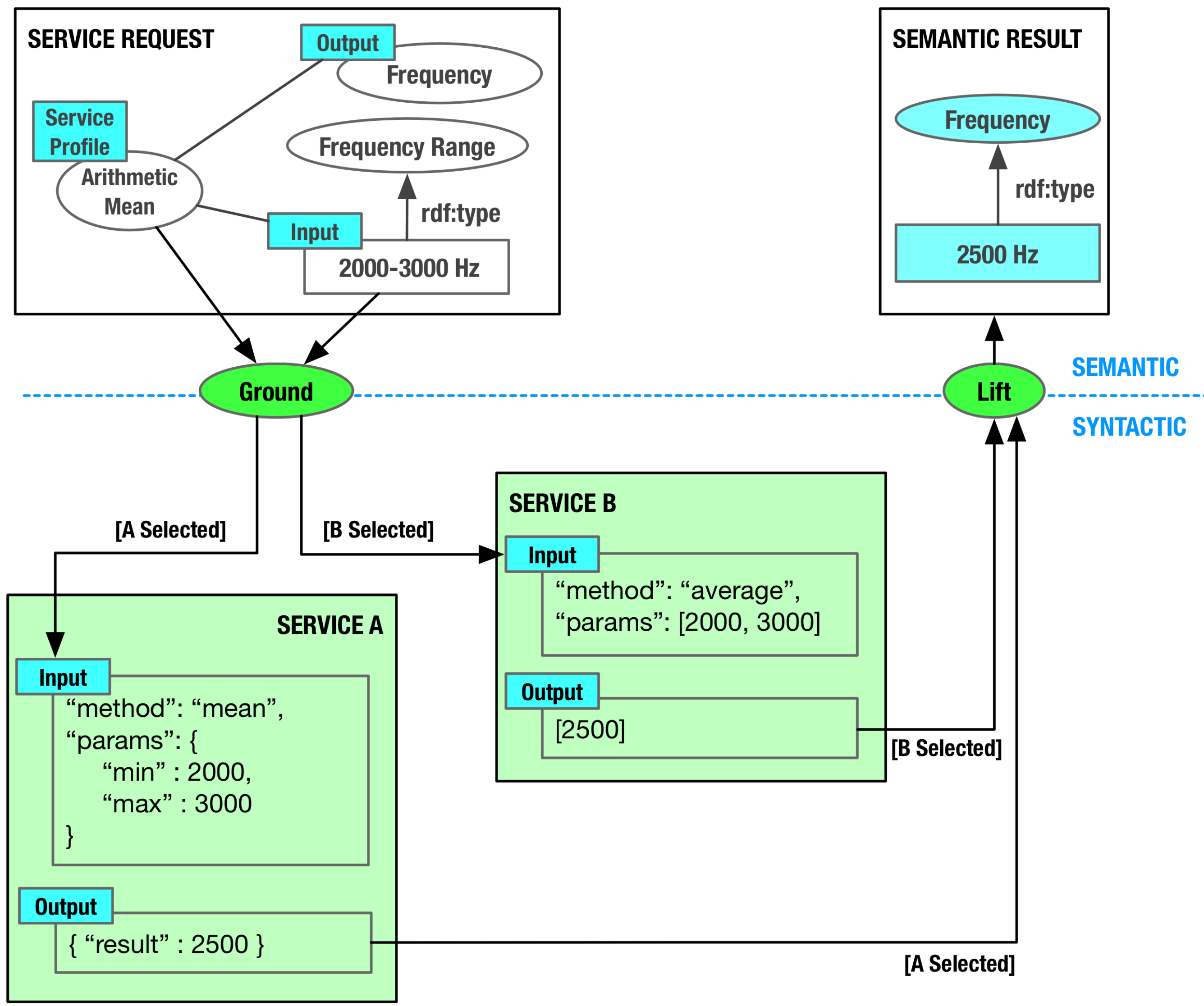
```
{ "jsonrpc": "2.0",
  "error": {
            "code": -32600,
            "message": "Invalid Request»
   },
  "id": null
}
```

Semantic Grounding and Lifting

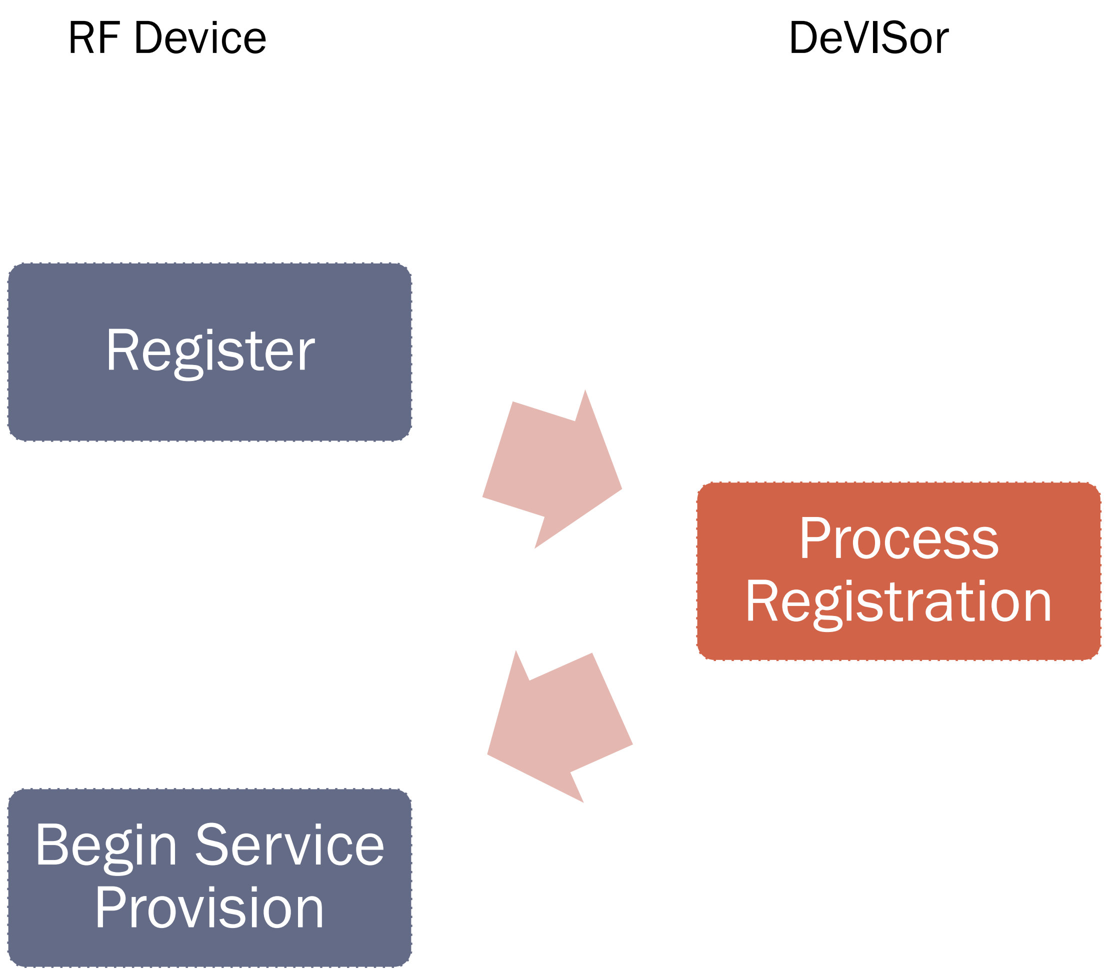
- The links between the abstract and the concrete
  - Grounding:
    - Convert abstract, ontological terms into concrete executable service invocation
  - Lifting:
    - Convert device-specific data structures to a common ontological representation that can be processed automatically
- Grounding/Lifting is specific to each device and must be provided during device registration
  - DeVISor uses this information when processing requests/results



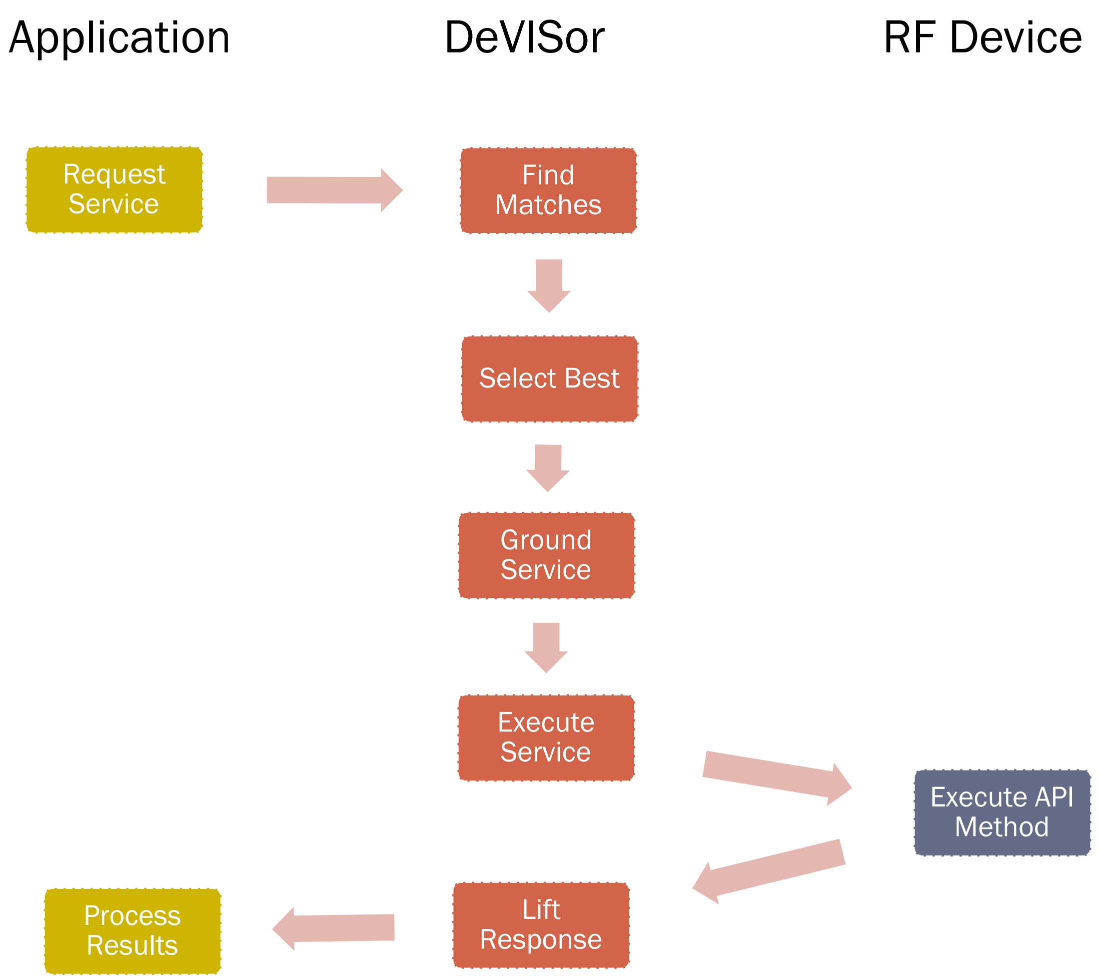
Semantic Grounding – Example



Concept of Operations – Device Registration



Concept of Operations – Service Request

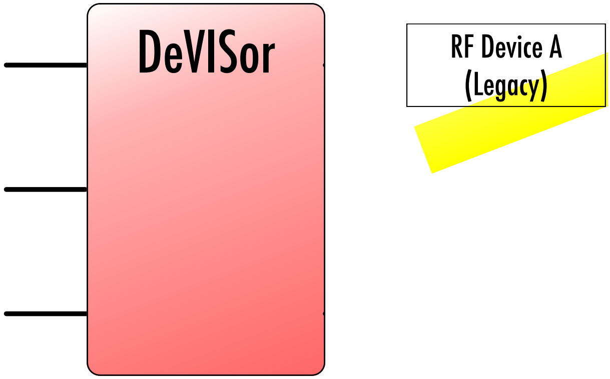


Key Benefits

- Devices:
  - Can have arbitrary API's to access their services – as long as they are available via JSON-RPC and semantically annotated during registration
  - Register services in terms of a common ontology – do not need to provide additional documentation
  - Can implement their services in virtually any programming language – JSON-RPC is a language-agnostic message-based mechanism
  - Can dynamically define new terms to provide future capabilities

- Applications:
  - Are developed independently of available devices
  - Do not need any knowledge of the API's
  - Formulate requests purely in terms of the common ontology
  - Know how to process results because they specify the expected output in ontological terms
  - Can easily combine the results with other facts expressed in the same ontological foundation

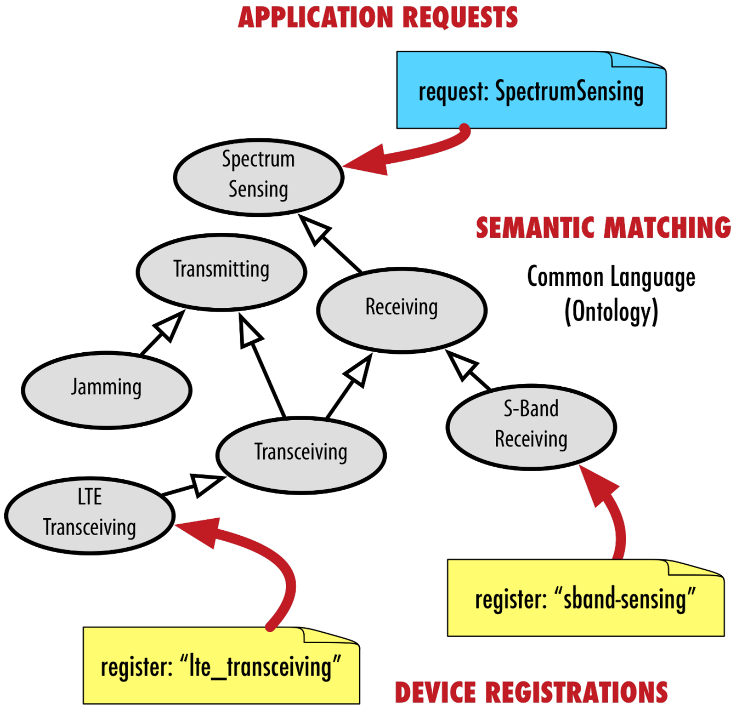
- Middleware (DeVISor):
  - Is agnostic of the domain terms, operates on a high-level plane of service requests and provisions
  - Does not need to be recoded to accommodate new domain terms (e.g. new jamming technique)



Minimal Requirements

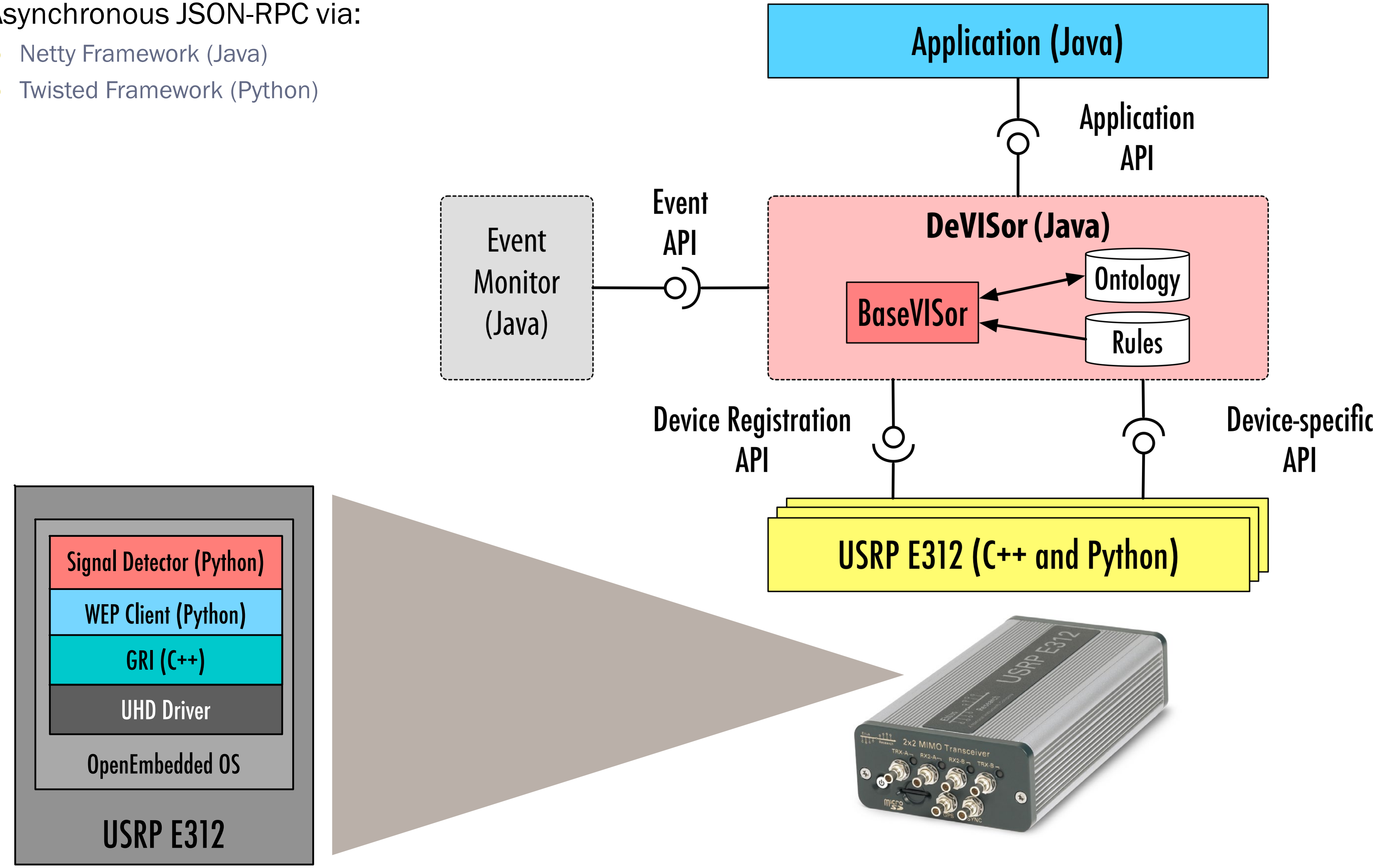
- Applications:
  - Use JSON-RPC to submit service requests
  - Express the requests in a common language (ontology)

- Devices:
  - Implement a single JSON-RPC method for registration
  - Express capabilities and map radio functions to the common language (ontology) as part of the registration
  - Provide access to radio functions via JSON-RPC interface

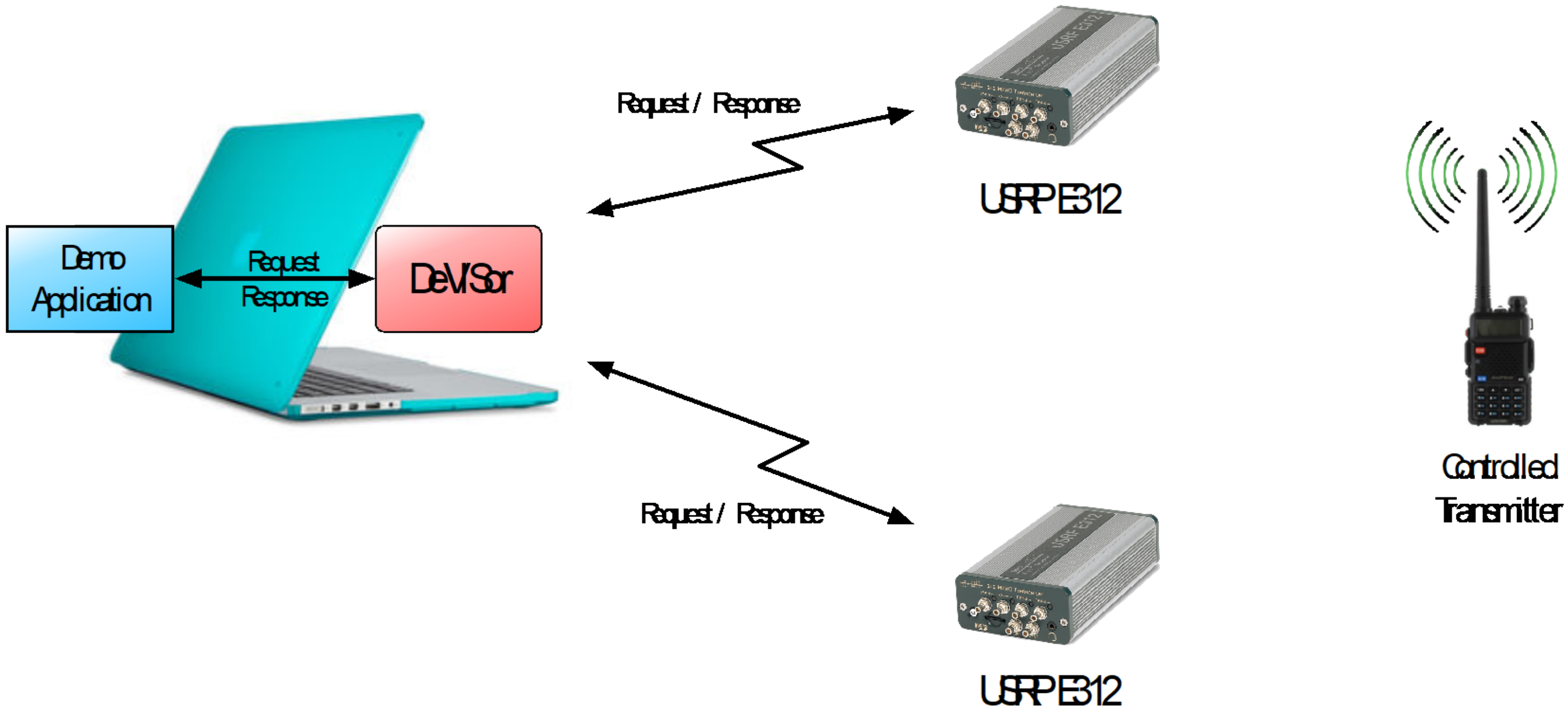


Implementation

- Asynchronous JSON-RPC via:
  - Netty Framework (Java)
  - Twisted Framework (Python)



Demo Setup



Demo Application
Request
Response
DeVISor
Request / Response
USRP E312
Request / Response
USRP E312
Controlled Transmitter

Thank You!

VISTOLOGY, INC.

HTTP://WWW.VISTOLOGY.COM

+1 (508) 788-5088

Mitch Kokar, President
mkokar@vistology.com

Jakub Moskal, CTO
jmoskal@vistology.com